ARMY RESEARCH LABORATORY

Early Results From the Porting of the Computational Fluid Dynamics Code, F3D, to the Silicon Graphics Power Challenge

by Daniel M. Pressel

ARL-TR-1562

December 1997

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-1562 **December 1997**

Early Results From the Porting of the Computational Fluid Dynamics Code, F3D, to the Silicon Graphics Power Challenge

Daniel M. Pressel
Corporate Information and Computing Center

Abstract

This report briefly discusses some of the issues involved in porting a computationally intensive code from a Cray vector supercomputer to the Silicon Graphics (SGI) Power Challenge (75-MHz Mips R8000-based shared memory Symmetric Multiprocessor). Additionally, some evidence is presented to indicate that similar results should be expected when taking this code to comparable boxes from other vendors (e.g., Digital Equipment, Convex, HP, or SUN). Performance results are also discussed. This discussion deals with both the highly successful nature of these results in terms of absolute levels of performance and the even more impressive results in terms of cost-effective performance. This report concludes with some thoughts on the future of this project.

Acknowledgments

The author wishes to express his gratitude to Mark Williams, formerly of the U.S. Army Research Laboratory (ARL), and Jim McElvaney of Silicon Graphics Inc. (SGI) for their assistance in providing the resources necessary to carry out this project. Additionally, he wishes to thank Karen Heavey, Jubaraj Sahu, James Collins, Walter Sturek, and Charles Nietubicz for their contributions to the success of this project. Finally, the author wishes to express his gratitude for the patience and support exhibited by his colleagues and the users of the computer resources at ARL, U.S. Army Chemical-Biological Defense Command (CBDCOM), Air Force Aeronautical Systems Center, Naval Command, Control and Ocean Surveillance Center, and elsewhere, while he carried out this study and prior studies that made this work possible.

This work was made possible through a grant of computer time by the Department of Defense (DOD) High-Performance Computing Modernization Program. Additionally, it was funded as part of the Common High-Performance Computing Software Support Initiative administered by the DOD High-Performance Computing Modernization Program.

INTENTIONALLY LEFT BLANK.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

INTENTIONALLY LEFT BLANK.

1. Overview

This project began as a result of an effort to run a large-memory (1.5 GB), computationally intensive job on a Silicon Graphics (SGI) Power Challenge. The code selected for this effort was an implicit 3-D computational fluid dynamics solver known as F3D.[*] The test case required a bit under 9 **CPU**[†] minutes to run on a Cray C90 when using one processor (a bit over 10 **CPU** minutes if run as an out-of-core solver on the same hardware, when using the solid-state disk to hold the data not currently resident in the main memory).[**] Attempts to run the in-core solver version of F3D on an SGI Power Challenge (75 MHz using just one processor), required over 5 hours of **CPU time**. The intent of this project was to determine the reasons and, if possible, the potential solutions for this poor level of performance. The most troubling of these causes was the possibility that the SGI Power Challenge and, by inference, other **RISC**-based shared memory **SMPs** were simply incapable of efficiently running large, computationally intensive problems.

Upon further investigation, several conclusions were reached.

(1) There were no fundamental flaws in the design of the SGI Power Challenge that would make it impossible to port this code to this machine.

(2) There was no inherent reason to believe that the algorithms used in this program were poorly suited to run on this machine. Quite the contrary, they looked most promising, both from the standpoint of serial efficiency and for their potential to be parallelized using loop-level parallelism (which seems to be a natural programming model for this class of machines).

---

[*] Sahu, J., and J. L. Steger. "Numerical Simulation of Transonic Flows." *International Journal for Numerical Methods in Fluids*, vol. 10, no. 8, pp. 855–873, 1990.

[†] Note: All items in **BOLD** type are defined in the Glossary.

[**] The test case only involved computing 10 time steps. Production runs frequently involve processing hundreds or even thousand of time steps and generally take many hours to finish.

(3) As currently written, the program showed almost no **locality of reference**, which easily explained why any architecture based on a hierarchical memory system would have trouble running this code. Fortunately, there was strong evidence that implementation-level tuning would be sufficient to substantially improve upon this situation.

(4) Many of the optimizations performed in an effort to obtain high levels of **vectorization** of this code were incompatible with the efficient operation of the code on the SGI Power Challenge.

(5) The compilers on most, if not all, of today's **RISC**-based architectures were incapable of producing efficient code, even after the problems (numbers 3–4 discussed previously) were largely eliminated. While the traditional approach to this problem would have been to use a combination of library routines and custom written assembly code, it was found that the judicious use of a highly aggressive programming style could achieve the same results without sacrificing portability. This effort was strongly based on earlier research done by the author.*

As a result of these efforts (numbers 3–5 previously mentioned), the tuned code is able to run the specified problem (correctly) in just under 30 minutes when using one processor of the SGI Power Challenge (a bit over 4 of the 30 minutes is required for startup and termination costs, mostly associated with the use of formatted I/O, the comparable number of the Cray C90 is about 100 s). When the run is parallelized across 12 processors, the run time drops to 6 1/2 minutes (roughly 3 1/2 minutes for startup and termination costs). Additionally, tests on an earlier version of this code indicate that there are substantial additional gains in performance from using more than 12 processors. (Unfortunately, the current configurations of the Power Challenge Array at the U.S. Army Research Laboratory (ARL) makes it difficult to repeat these measurements on the latest version of the code.)

* Pressel, D. M. Unpublished Research. U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.

Since the specified problem involves running just 10 time-steps, the author feels that the best comparisons are made with the startup and termination costs removed. Based on this practice, the serial code is now running 11 1/2 times faster than the original code. The parallelized code, when using 12 processors, is running 100 times faster than the original code. Additionally, the parallelized code using 12 processors is running more than two times faster than the original code ran on 1 processor of a Cray C90 (prior experience indicates that using 18 processors would almost certainly increase this number to a factor of 3). Note that these results were achieved without in any way changing the convergence properties of this code (e.g., through the use of an explicit algorithm, domain decomposition techniques or the introduction of other types of approximations as a way to increase the parallelizability of the code). This is very important, since it means that the code will require the same number of time steps to converge, thereby establishing the validity of these comparisons.

2. Source of the Problem

Many vendors, including SGI, have produced **RISC**-based **SMPs** with some variant of shared memory/virtual shared memory. Invariably, they claim/seem to be claiming that since people have been writing programs for shared memory environments for years, it should be trivial to port their programs to the new machines.[*][†][**][††] Also, invariably, things are far from being this simple. So, what are the issues? The following lists some of the more important issues.

(1) The "simple porting" theory is based on the premise that the **PRAM** model, which is taught in some computer science departments, is an accurate representation of the current generation of machines. The problem with this assumption is that the **PRAM** model makes a simplifying assumption of a large, flat memory system. While this might be a reasonable

[*] *The Advent of Power Computing*. Silicon Graphics Inc., 1994.

[†] *Power Learn: Achieving High Performance and Parallelism With Silicon Graphics Power Challenge Systems.* Silicon Graphics Inc., 1995.

[**] *Exemplar Architecture*. Convex, 1993.

[††] *KSR1 Principles of Operation.* Kendall Square Research, 1992.

approximation for the memory system of a Cray vector supercomputer or some of the earlier products from Sequent (and others), in no way is it an adequate representation of the current generation of machines.

(2) Traditionally, there have been two classes of shared memory **SMPs**. Unfortunately, for the following reasons, neither class of machine was used to produce programs that are well suited for use on systems such as the Power Challenge.

   (a) Those made by Sequent and its competitors were composed of processors that were generally considered to be too slow for use on computationally intensive jobs. As a result, there are relatively few computationally intensive codes that have been tuned for these machines, and most of the codes that do exist are for jobs that were difficult to vectorize.

   (b) Those made by Cray Research, and to a lesser extent, its competitors were based on a large, flat, multibanked memory system, relying on the vector pipelining to hide the latency of the memory system, while supporting a high level of performance. Traditionally, codes written for these machines were optimized for peak levels of performance of a single vector processor, and only rarely was any effort made to use multiple processors for a single job. Additionally, the theoretical underpinnings of these machines are dramatically different than that for the current generation of shared memory **SMPs**.

(3) Another selling point for the current generation of shared memory **SMPs** is that they are a natural extension of the **RISC** architectures used in the workstations found on the desks of most scientists and engineers involved in High-Performance Computing. While this statement is true, experience teaches that only a modest percentage of the programs run on these workstations have been tuned for them. In general, it was considered to be easier and more cost-effective to just buy a faster machine or use a departmental server (e.g., a large DEC VAX) or a supercomputer from Cray or one of its competitors, than to tune/retune

one's program. As a result, there seems to be little interest in or support for efforts to determine what programming styles were best suited for use with **RISC**-based workstations.

So, to summarize the problem, potentially many of the current generation of shared memory **SMPs** are powerful enough to compete with traditional vector supercomputers. Additionally, the claims that these machines are natural extensions to existing machines and should therefore be easy to port code to is also true; or a least it would be true if there were a significant body of existing codes that had already been optimized for **RISC**-based shared memory **SMPs** with a hierarchical memory system.[*][†][**][††] Unfortunately, few such codes currently exist, and there appears to be little research in this area.

3. Solution

While it would be tempting to conclude that the solution is to parallelize the code across as many processors as possible (possibly even using multiple Power Challenges). Based on two observations, doubts were expressed about this methodology. First, given the unbelievably poor performance of the code when using one processor of the Power Challenge, it was unlikely that this approach would be cost-effective. Secondly, since the parallel efficiency of most codes tends to decrease as the number of processors increases, it was far from clear if this approach would even work. This view was reinforced by the observation that the current algorithm supported only a limited amount of parallelism.

Based on this analysis, the author set about the rather tedious job of retuning the code for a substantially improved level of single processor performance. Early efforts were guided by profiling reports that indicated that most of the time was spent in only a few routines, and that several of the

[*] *The Advent of Power Computing.* Silicon Graphics Inc., 1994.

[†] *Power Learn: Achieving High Performance and Parallelism With Silicon Graphics Power Challenge Systems.* Silicon Graphics Inc., 1995.

[**] *Exemplar Architecture.* Convex, 1993.

[††] *KSR1 Principles of Operation.* Kendall Square Research, 1992.

most expensive routines were spending upward of 90% of their time processing **cache and TLB misses** (part of the memory management system). Early efforts in this area were aimed at increasing the use of Stride-1 memory access patterns (accessing array elements in the same order as they are stored in memory). This was accomplished by changing the order of the indices for many of the larger arrays. Additionally, limited use was made of large scratch arrays and transposes in an attempt to isolate code that was accessing memory in other than a Stride-1 pattern in subroutines that could be easily blocked (another method for increasing the **locality of reference**).

While these changes were, in principle, trivial to implement, the sheer number of changes required made this a very time-consuming and painful undertaking. The result of these changes was to reduce the run time to under 2 hours. Additional analysis showed that a significant number of scratch arrays stored an entire plane of data at one time. This approach had been adopted as a result of some of the efforts to maintain the **vectorizability** of the code. Since that was no longer a concern for this project, and since the size of these arrays made them poorly suited for use on a computer with a hierarchical memory system, the decision was made that certain key loops should be modified so that they processed data a slice at a time, rather than processing an entire plane of data at a time. It was demonstrated that these changes could be made without fundamentally changing the algorithm, and the effect was to reduce the size of many of the scratch arrays to the point that they fit comfortably in the secondary **cache.**[*] The result of this set of transformations, as well as some rather simple transformations designed to reduce the amount of unnecessary data motion associated with switching between zones, further reduced the run time to under 1 hour and 30 minutes.

While it might have been practical at this point to switch from the serial tuning of the code to parallelizing the code, it was felt that substantial gains could still be realized by using some additional techniques.

[*] For some time, it was not clear what the optimal slice size should be. The current implementation uses an adaptive technique based on the specifics of the computer being used, the problem size, and the number of processors being used. It should be noted that this is an area of continuing research.

(1) Eliminate unnecessary calculations and/or data motion whose primary purpose seems to be to maintain the **vectorizability** of the code. While this technique frequently reduces the readability of the code, in many cases, it can improve the performance by nearly an additional factor of 2. (This is based on the metric of primary interest to the end user—the run time—rather than maximizing the number of floating-point operations per second.)

(2) Perform a variety of fairly standard optimizations (e.g., loop unrolling) in an attempt to identify additional opportunities for optimizing the code. In many cases, the result was the identification of new tuning opportunities that few, if any, compilers were likely to find on their own.

(3) The author has observed that while many modern optimizing compilers for current **RISC**-based platforms are capable of performing an interesting array of optimizations, they tend to give up when the complexity of a loop crosses a certain threshold. The result can be a very sharp dropoff in the performance of the code that is produced. The problem is that many of the previously mentioned techniques have the tendency to produce large, but not necessarily complicated, loops. Unfortunately, the size of these loops will, in general, cause the optimization routines to give up. As a result, the author has developed a series of techniques for manual software pipelining C and FORTRAN routines. In general, these techniques can produce additional improvements in performance by a factor of 2–5 (or more). The unfortunate side effect is a dramatic reduction in the readability/maintainability of the code.* Therefore, it is unwise to use these techniques throughout the code. However, applying them in a judicious manner, one can still achieve impressive incremental gains in performance. It should also be noted that while these techniques have been shown to be of value for virtually all **RISC** processors, highly aggressive processors, such as the MIPS R8000 processor used in the Power Challenge, are likely to benefit the most from this step. Additionally, since this step requires the programmer to have some concept of the architectural characteristics of the target processor, it is likely that the benefit of using these

* A former colleague of the author referred to this technique as Writing Assembly Code masquerading as C or FORTRAN.

techniques vis-a-vis other processors may not be quite as large. Initial tests conducted on systems based on the SUN/TI SuperSPARC, HP PA-RISC 7100, and MIPS R4400 processors were all very encouraging (within the limits of the individual system designs).

The net result of all of these transformations was the eventual reduction in the single-processor run time to just under 30 minutes. It was also noted that almost all of these optimizations were shown to be beneficial to the parallelized code (using the c$doacross directive to implement loop level parallelism). By continuing to pay attention to details, 12 processor run times of 6 minutes and 30 seconds (of which 3 minutes and 30 seconds were startup and termination costs) were achieved. Clearly, if one has the time to spend, applying these techniques to a production code such as F3D, can be viewed as a major success. Additionally, since none of these techniques required any knowledge or understanding of the algorithms used in this program, they should be widely applicable. In fact, many of the more aggressive techniques were originally developed on previous projects.*

Figures 1 and 2 provide performance results for this work for two different-sized problems.† These figures compare two different versions of the code running on a Cray C90 vs. the new RISC-optimized version of the code running on a 75-MHz R8000-based Power Challenge. Several interesting things to note on these charts are:

(1) One can achieve a modest, but noticeable improvement in performance on the C90 simply by running everything in-core. Unfortunately, the large memory requirements for that optimization have discouraged its use.

(2) These charts are based on runs for a limited number of time steps. As such, the reader will get a fairer view of the relative performance of the code by subtracting the start-up and termination costs. This process results in what is referred to as the adjusted speed and is a good estimate of the performance of the code when run for large numbers of time steps.

* Pressel, D. M. Unpublished Research. U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.

† Actually, they are the same problem; just the grid size was changed.

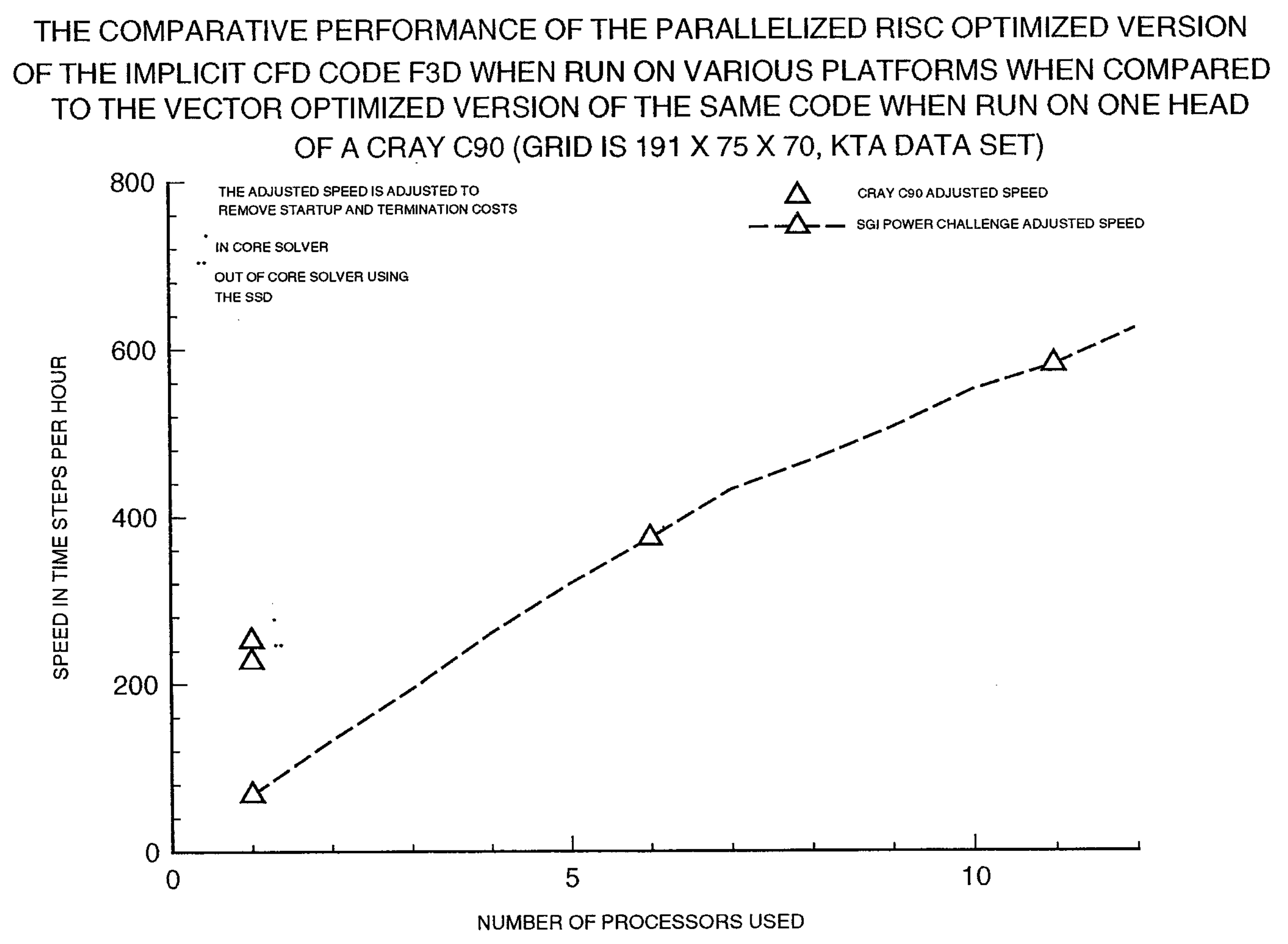

Figure 1. The Performance of the 1-Million Grid Point KTA Test Case on One Processor of a Cray C90 and a 12-Processor SGI Power Challenge (75 MHz).

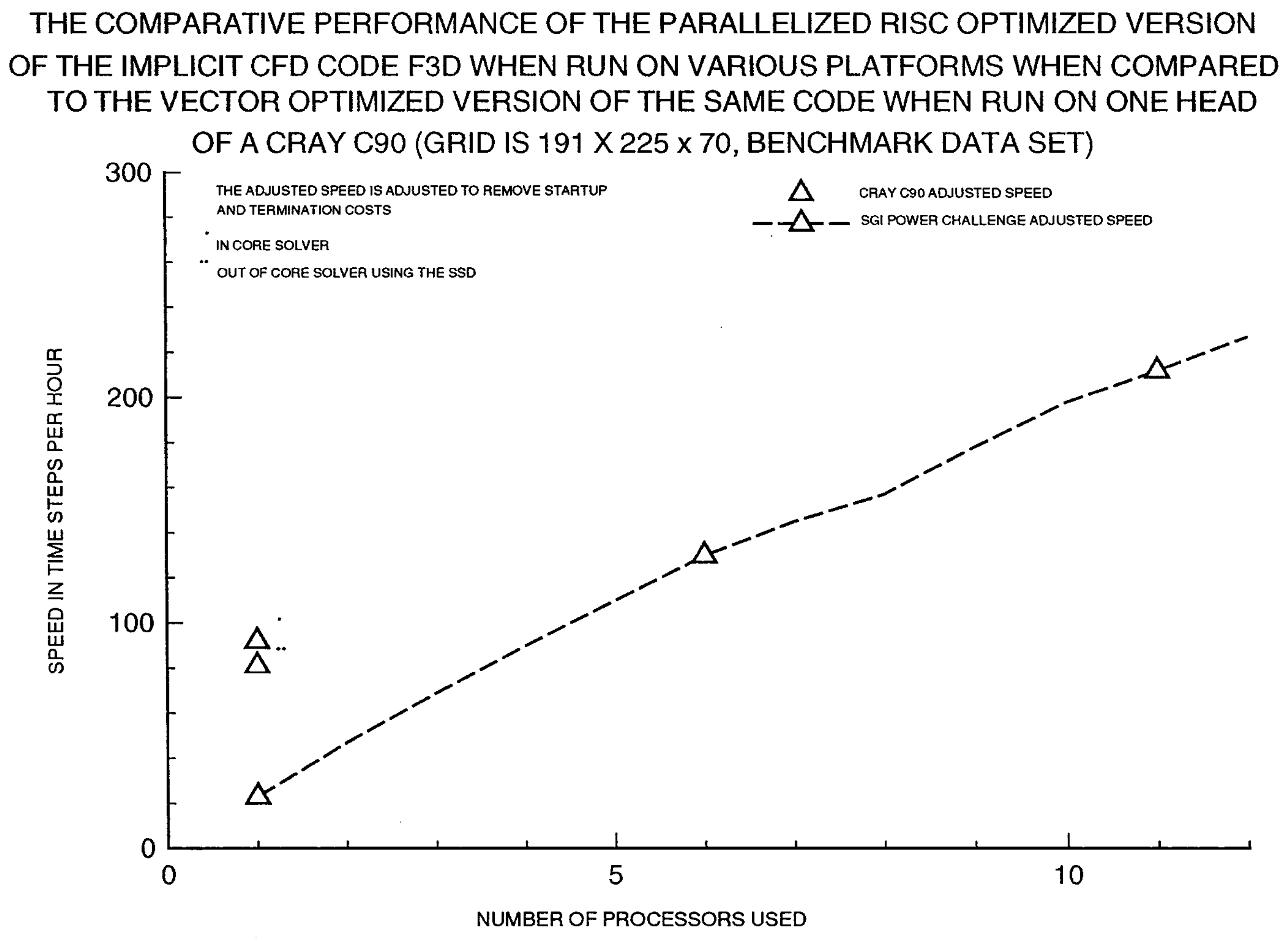

Figure 2. The Performance of the 3-Million Grid Point Benchmark Test Case on One Processor of a Cray C90 and a 12-Processor SGI Power Challenge (75 MHz).

(3) Since the Cray versions of the code are considered to be efficient, the speed in time steps per hour is the most useful measure of performance. (Normally, this is the only measure of performance that the user cares about.)

(4) In discussions with other researchers, the author has found it interesting to note that he seems to be getting roughly twice the per processor performance on the SGI Power Challenge than what most of the others are reporting for small numbers of processors.[*][†][**][††] For larger numbers of processors, this difference frequently increases to a factor of 3–4.[***]

(5) It is also important to note that there is no obvious dropoff in performance when one goes to a larger problem size.

Figures 3 and 4 were prepared by Karen Heavey and Jubaraj Sahu, U.S. Army Research Laboratory (ARL), as part of the code validation effort. Figure 3 shows the computed pressure contours obtained at 1,800 time steps using the Power Challenge and C90. Both results are practically identical. Figure 4 shows the circumferential surface pressure distributions at two selected longitudinal stations. Again, both Power Challenge and C90 results are virtually the same. In fact, both computed results, when overlaid, lie on top of each other. These results provide strong evidence that the tuned code is behaving properly.

4. Future Plans

At the present time, several additional efforts relating to this code are underway. Some of these are as follows:

[*] Wallcraft, A. J. Private communication. 1997 DOD HPC User's Group Meeting, 1996.

[†] Sirbaugh, J. Private communication. 1997 DOD HPC User's Group Meeting, 1996.

[**] Smarr, L. Presentation at 1996 DOD HPC User's Group Meeting, 1996.

[††] Wallcraft, A. J. Presentation. 1997 DOD HPC User's Group Meeting, 1997.

[***] These comparisons were generally made in terms of MFLOPS/second where the operation count was measured using the hardware performance monitor for the Cray C90 when running the vector-optimized version of the code. This seems to be the most objective technique when comparing the performance of dissimilar programs.

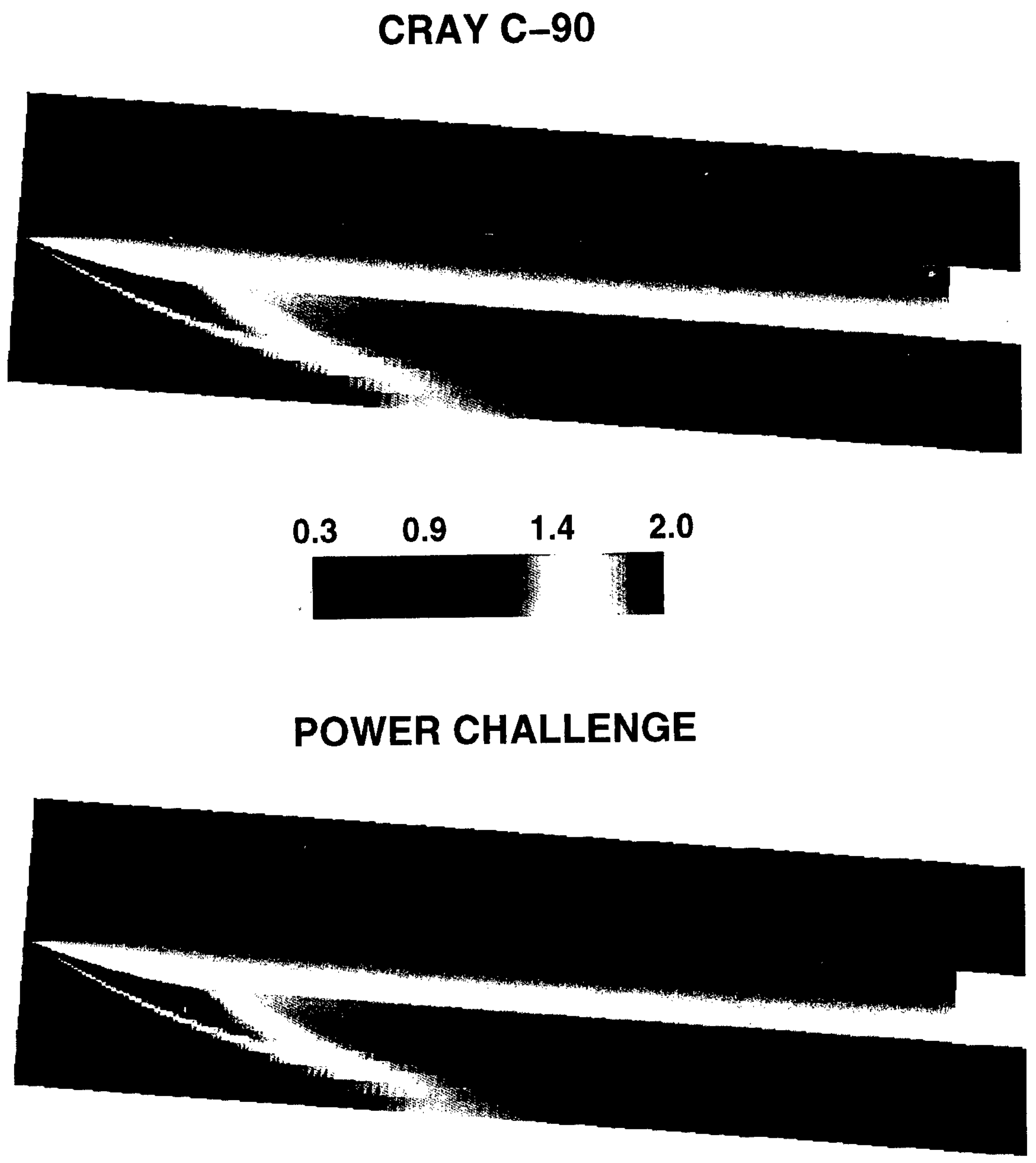

Figure 3. Computed Pressure Contours at 1,800 Time Steps.

(1) Marek Behr, Ph.D., of the Army High-Performance Computer Research Center has been porting the same code to the Cray T3D and other traditional Distributed Memory **RISC**-based **MIMD MPPs**. Serial optimizations developed for this effort have been provided to Marek Behr and adapted for use in his work.

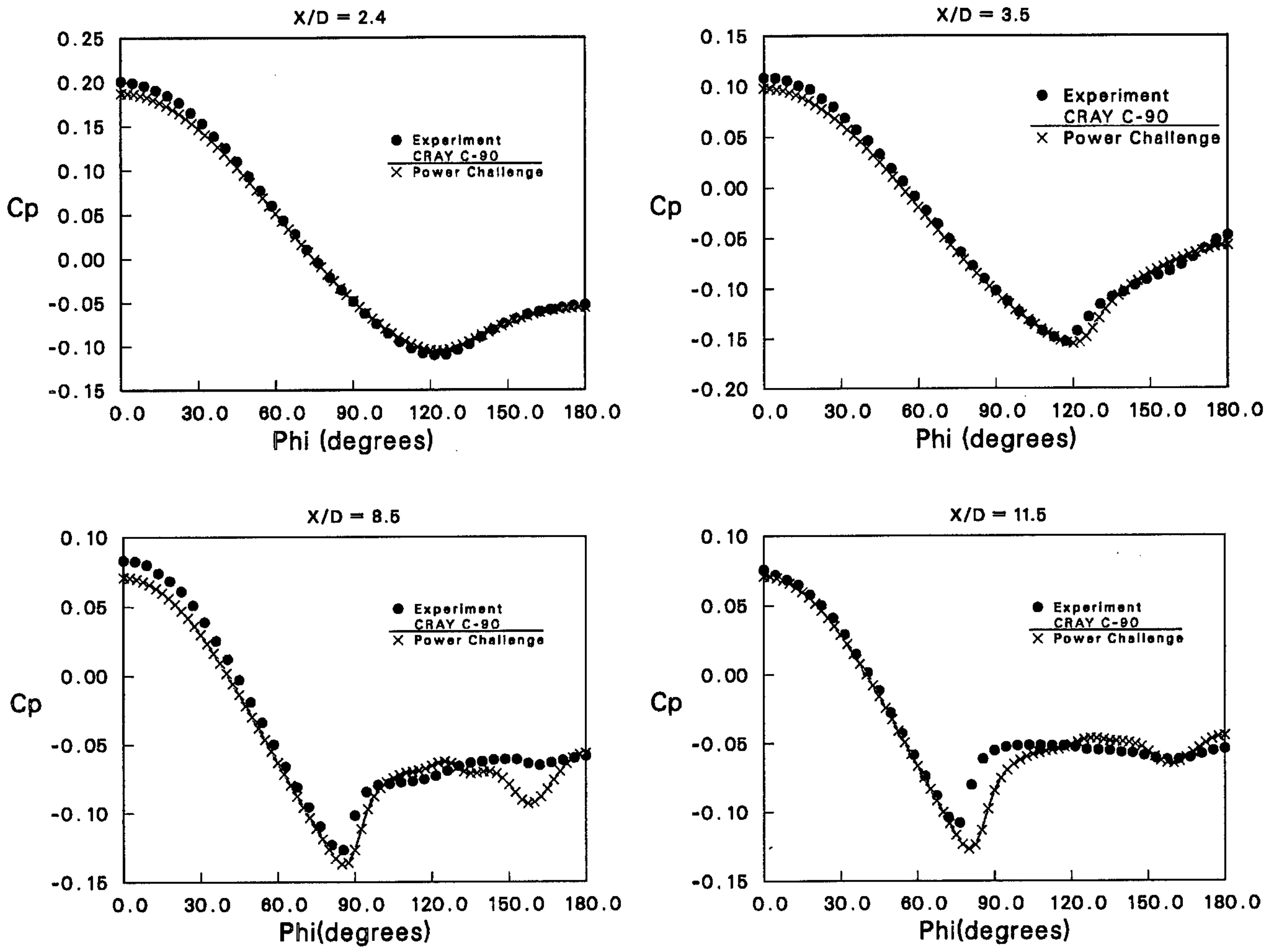

Figure 4. Circumferential Pressure Plots at 1,800 Time Steps.

(2) James Collins, Ph.D., of ARL is currently adding several additional modules to this code, including an implementation of the CHIMERA technique.

(3) James Collins, Jubaraj Sahu, Ph.D., also of ARL and others are beginning work on a formal plan to revalidate the tuned code.

(4) Through additional investigations, more techniques into ways to improve the performance of this code when using larger numbers of processors on a shared memory **SMP** such as the SGI R10000-based Power Challenge or on a virtual shared memory **SMP** such as the Convex Exemplar family of computers. It is also expected that tests will be conducted to identify, and if necessary, correct any problems that might occur when the problem is scaled to even larger sizes. Unfortunately, only recently have systems with sufficient memory and address space to handle problems that are significantly larger than those used in this study come online. It is expected that results using these systems will be discussed in future reports.

(5) Some of the routines currently in the F3D code were neither optimized (other than to the extent necessary to maintain their validity) nor parallelized. While many of these routines are so fast that there was no need to tune them, a significant percentage of the untuned routines are computationally intensive, but were not used in processing the benchmark case. Eventually, it would be desirable if some effort were made to either improve the speed of these routines or delete them from the standard distribution.

5. Conclusions

It is clear that some computationally intensive codes can be tuned so that a meaningful range of problem sizes can be run with an acceptable level of performance on the current generation of **RISC**-based shared memory **SMPs**. Having said that, it is also clear that transitioning code to these machines will be far from the plug and play process many potential users are hoping for. Additionally, our experience is that when running well-tuned code on the Power Challenge, the price-to-performance can be favorable.[*] If other researchers are able to obtain similar results, this may have beneficial long-term consequences.

[*] This is based on results reported in this report and list price data supplied by SGI.

Now, let us look at the way technology is advancing. High-end **RISC** processors are getting significantly faster and are available in both desktop workstations and shared memory **SMPs** such as the Power Challenge. Additionaly, memory prices for these machines have dropped to the point where it is cost-effective to put hundreds of megabytes of memory in a workstation and several gigabytes of memory in the **SMPs**. As such, it is reasonable to expect that, as time progresses, it will be practical to run even larger jobs on the **SMPs** and that many of the current jobs may actually be capable of migrating to well-equipped workstations.

It appears as though a major stumbling block to this scenario becoming a reality is the lack of well-tuned codes for this new class of machines. Therefore, what we need is to identify which codes are good candidates for transitioning to this new class of machines. Additionally, since it takes time to carry out this effort, we suggest, in part, basing the selection criteria on the predicted performance of the next-generation of **SMPs**. It is our experience that the techniques we used have a substantial benefit on multiple platforms and should therefore still be of significant value in succeeding generations of hardware.

INTENTIONALLY LEFT BLANK.

Glossary

Cache - One way in which high-speed memory is used by modern computer architectures.

Cache and TLB Miss - **Caches** and **TLB**s are used to store frequently needed information so that the processor can avoid accessing main memory, which can take up to 100 times as long. A miss occurs when the required information is not already present in the **cache** or the **TLB** (as appropriate) and must first be fetched from main memory.

CPU - Central processing unit.

CPU Time - The total amount of time that the **CPU** actually spends executing a job. Note: it does not matter if one uses 10 **CPU**s for 1 minute, or 1 **CPU** for 10 minutes. In either case, one has used 10 **CPU** minutes.

Locality of Reference - This refers to a property exhibited by many programs, where either (1) the same memory location is accessed repeatedly over a short period of time, and/or (2) a continuous range of memory locations is accessed (usually in order) over a period of time.

MIMD - Multiple instruction/multiple data. This refers to a class of parallel-processor-based computers.

MPP - Massively parallel processor. Generally refers to computers composed of 100 or more processors.

PRAM - Parallel Random Access Machine. A commonly used model for the study of parallel computer architectures and programs.

RISC - Reduced instruction set computer.

SMP - Symmetric multiprocessor.

TLB - Translation lookaside buffer (part of the memory management system).

Vectorizability - The potential for a program to run effectively on a vector processor such as a Cray C90.

Vectorization - The creation of an executable program that will take advantage of a vector processor's hardware.

Wall Clock Time - the total elapsed time between when a job starts to run and when it finishes. This is usually the only number the user will care about, but may vary over a wide range when a system is being shared with other jobs.

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ DENNIS SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	DPTY ASSIST SCY FOR R&T SARD TT F MILTON RM 3EA79 THE PENTAGON WASHINGTON DC 20310-0103
1	OSD OUSD(A&T)/ODDDR&E(R) J LUPO THE PENTAGON WASHINGTON DC 20301-7100
1	CECOM SP & TRRSTRL COMMCTN DIV AMSEL RD ST MC M H SOICHER FT MONMOUTH NJ 07703-5203
1	PRIN DPTY FOR TCHNLGY HQ US ARMY MATCOM AMCDCG T M FISETTE 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	DPTY CG FOR RDE HQ US ARMY MATCOM AMCRD BG BEAUCHAMP 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	GPS JOINT PROG OFC DIR COL J CLAY 2435 VELA WAY STE 1613 LOS ANGELES AFB CA 90245-5500
1	ELECTRONIC SYS DIV DIR CECOM RDEC J NIEMELA FT MONMOUTH NJ 07703
3	DARPA L STOTTS J PENNELLA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MDN A MAJ DON ENGEN THAYER HALL WEST POINT NY 10996-1786
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AL TP 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AL TA 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	<u>ABERDEEN PROVING GROUND</u>
4	DIR USARL AMSRL CI LP (305)

NO. OF
COPIES ORGANIZATION

1 USAE WATERWAYS
EXPERIMENT STATION
CEWES HV C
J P HOLLAND
3909 HALLS FERRY ROAD
VICKSBURG MS 39180-6199

1 DIR USARL
AMSRL PS E
B S PERLMAN
FORT MONMOUTH NJ 07703

1 NCCOSC
RDT&E DIVISION NRaD
CODE 404
R A WASILAUSKY
53570 SILVERGATE AVE
SAN DIEGO CA 92152-5180

1 NCCOSC
RDT&E DIVISION NRaD
CODE 7601T
K BROMLEY
5180 SILVERGATE AVE
SAN DIEGO CA 92152-5180

1 ARMY HIGH PERF COMPUTING RSRCH CTR
M BEHR
SUITE 101
1100 WASHINGTON AVE SOUTH
MINNEAPOLIS MN 55415

1 ARMY HIGH PERF COMPUTING RSRCH CTR
T TEZDUYAR
1200 WASHINGTON AVE SOUTH
MINNEAPOLIS MN 55415

1 ARMY HIGH PERF COMPUTING RSRCH CTR
B BRYAN
1200 WASHINGTON AVE SOUTH
MINNEAPOLIS MN 55415

1 ARMY HIGH PERF COMPUTING RSRCH CTR
G V CANDLER
1200 WASHINGTON AVE SOUTH
MINNEAPOLIS MN 55415

1 ARMY AEROFLIGHT DYNAMICS DIR
R MEAKIN
M/S 258-1
MOFFETT FIELD CA 94035-1000

NO. OF
COPIES ORGANIZATION

1 WL/FIMC BLDG 450
S SCHERR
2645 FIFTH ST SUITE 7
WPAFB OH 45433-7913

1 COMMANDER
CODE C2892
C HOUSH
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555

1 NAVAL SURFACE WARFARE CENTER
CODE B44
A B WARDLAW
SILVER SPRING MD 20903-5640

1 NAVAL RESEARCH LAB
CODE 6400
J BORIS
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-5344

1 NAVAL RESEARCH LAB
CODE 6410
R RAMAMURTI
WASHINGTON DC 20375-5344

1 NAVAL RESEARCH LAB
HEAD OCEAN DYNAMICS AND
PREDICTION BR
CODE 7320
J W MCCAFFREY JR
STENNIS SPACE CENTER MS 39529

1 WL/FIMC BLDG 450
B STRANG
2645 FIFTH ST SUITE 7
WRIGHT-PATTERSON AIR FORCE BASE OH
45433-7913

1 WL/FIM
J J S SHANG
2645 FIFTH STREET STE 6
WRIGHT-PATTERSON AIR FORCE BASE OH
45433-7912

1 US AIR FORCE PHILIPS LAB
OLAC PL/RKFE
CPT S G WIERSCHKE
10 EAST SATURN BLVD
EDWARDS AIR FORCE BASE CA
93524-7680

NO. OF COPIES	ORGANIZATION
1	US AIR FORCE ROME LAB RL/OCTS R W LINDERMAN GRIFFISS AIR FORCE BASE NY 13441-5700
1	PROJECT MANAGER CHSSI R FOSTER 1110 N GLEVE ROAD SUITE 650 ARLINGTON VA 22201
1	DIRECTOR DEPT OF ASTRONOMY 356 PHYSICS BLDG 116 CHURCH STREET SE MINNEAPOLIS MN 55455

NO. OF COPIES	ORGANIZATION
	ABERDEEN PROVING GROUND
14	DIR USARL AMSRL CI HA C NIETUBICZ W STUREK AMSRL CI HC D PRESSEL J COLLINS D HISLEY C ZOLTANI J GROSH A PRESSLEY T KENDALL P DYKSTRA AMSRL WM BC H EDGE J SAHU K HEAVEY P WEINACHT

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Final, Jan 95 - Dec 96

4. TITLE AND SUBTITLE Early Results From the Porting of the Computational Fluid Dynamics Code, F3D, to the Silicon Graphics Power Challenge	5. FUNDING NUMBERS 61110ZH4800
6. AUTHOR(S) Daniel M. Pressel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: AMSRL-CI-HC Aberdeen Proving Ground, MD 21005-5067	8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-1562
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)	10.SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report briefly discusses some of the issues involved in porting a computationally intensive code from a Cray vector supercomputer to the Silicon Graphics (SGI) Power Challenge (75-MHz Mips R8000-based shared memory Symmetric Multiprocessor). Additionally, some evidence is presented to indicate that similar results should be expected when taking this code to comparable boxes from other vendors (e.g., Digital Equipment, Convex, HP, or SUN). Performance results are also discussed. This discussion deals with both the highly successful nature of these results in terms of absolute levels of performance and the even more impressive results in terms of cost-effective performance. This report concludes with some thoughts on the future of this project.

14. SUBJECT TERMS computational fluid dynamics, super computing, high performance computing	15. NUMBER OF PAGES 26
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-1562 ____ Date of Report December 1997

2. Date Report Received ____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) ____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) ____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. ____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) ____

CURRENT ADDRESS	Organization	
	Name	E-mail Name
	Street or P.O. Box No.	
	City, State, Zip Code	

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS	Organization
	Name
	Street or P.O. Box No.
	City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)